Lucky Beulla Muhoza
CS 338
Prof. Jeff Ondich

Sept 27^{th}, 2024

Being eve

- Intercepted for Diffie Hellman.

✓ $g = 7$ and $p = 97$
✓ Alice sent Bob 53 → A
✓ Bob sent Alice 82 → B

We know :

$A = g^{a} \bmod p$ a → random from Alice

$B = g^{b} \bmod p$ b → random from Bob

⟹

$K = g^{ab} \bmod p$
$= A^{b} \bmod p$ → for Bob
$= B^{a} \bmod p$ → for Alice

Alice

* $53 = 7^{a} \bmod 97$

Compute for a

```
IndentationError: expected an indented b
>>> for a in range (1,97):
...     if 7**a % 97 ==  53:
...             print (f'  a = {a}')
...
  a = 22
```

Bob

* $82 = 7^{b} \bmod 97$

Compute for b

```
>>> for b in range (1,97):
...     if 7**b % 97 == 82:
...             print (f'b={b}')
...
b=41
```

$K_A = B^a \bmod p$ $\qquad$ $K_B = A^b \bmod p$

$K_A = 82^{22} \bmod 97$ $\rightarrow K_A = K_B \leftarrow$ $K_B = 53^{41} \bmod 97$

$= 65$ $\qquad$ $= 65$

```
>>> 82**22 % 97
65
>>> 53**41 % 97
65
>>>
```

Alice and Bob's shared secret = $\underline{\underline{65}}$

* This process would have failed at the exponential calculations $g^a \bmod p$ and $g^b \bmod p$ for very large integers g and p since the basic exponentiation would significantly grow if these large numbers were involved, making the computing process impractical.

- Intercepted for RSA

Bob's public key :

$$\underbrace{(e\_Bob,\ n\_Bob)}_{P_{\text{public key}}} = (\underbrace{13}_{e},\ \underbrace{162991}_{n})$$

We need d :

$$e_B\, d_B \bmod \lambda(n_B) = 1$$

$$\hookrightarrow \lambda(n_B) = \mathrm{lcm}(p_B - 1,\ q_B - 1)$$

$$n_B = \underbrace{162991}_{p \times q}$$

Compute p and q

```
>>> import math
>>> n = 162991
>>> for p in range(2, math.isqrt(n) + 1):
...     if n % p == 0:
...             q = n // p
...             print(f"Factors of n are p = {p} and q = {q}")
...
Factors of n are p = 389 and q = 419
```

$p_B = 389$ $\qquad q_B = 419$

$$\lambda(n_B) = \mathrm{lcm}(388,\ 418)$$

```
>>> import math
>>> lcm = math.lcm (388, 418)
>>> print(f'lamda_n = {lcm}')
lamda_n = 81092
```

$$\lambda(n_B) = 81092$$

$e \cdot d \mod \lambda(n) = 1$

i.e $13 \cdot d \mod 81092 = 1$

```
>>> e = 13
>>> lamda_n = 81092
>>> for k in range (1, lamda_n):
...     if e * k % lamda_n == 1:
...             print(f'd = {k}')
...
d = 43665
```

We know that we can decrypt the ciphertext by

$$M = \underbrace{c}_{\text{ciphertext}}^{d} \mod n$$

thus :

```
>>> ciphertext = [17645, 100861, 96754, 160977, 120780, 90338, 130962, 74096,
... 128123, 25052, 119569, 39404, 6697, 82550, 126667, 151824,
... 80067, 75272, 72641, 43884, 5579, 29857, 33449, 46274,
... 59283, 109287, 22623, 84902, 6161, 109039, 75094, 56614,
... 13649, 120780, 133707, 66992, 128221]
>>> e=13
>>> d=43665
>>> n=162991
>>> plaintext = [pow(c, d, n) for c in ciphertext]
>>> print ('Plaintext: ', plaintext)
Plaintext:  [17509, 24946, 8258, 28514, 11296, 25448, 25955, 27424, 29800, 26995, 8303, 30068, 11808, 26740, 29808, 294
6991, 12064, 21349, 25888, 31073, 11296, 16748, 26979, 25902]
```

↳ plaintext numbers not ASCII values,

Alice might have encoded the message by encoding multiple characters together.

Plaintext result = base conversion

(base - 256 format)

↳ Aha!

```
>>> def base256_to_ascii(num):
...     chars = []
...     while num > 0:
...             chars.append(chr(num % 256))
...             num //= 256
...     return ''.join(reversed(chars))
...
>>> plaintext_nums = [17509, 24946, 8258, 28514, 11296, 25448, 25955, 27424, 29800, 26995,
...                     8303, 30068, 11808, 26740, 29808, 29498, 12079, 30583, 30510, 29557,
...                     29302, 25961, 27756, 24942, 25445, 30561, 29795, 26670, 26991, 12064,
...                     21349, 25888, 31073, 11296, 16748, 26979, 25902]
>>> decoded_message = ''.join(base256_to_ascii(num) for num in plaintext_nums)
>>> print ("Alice's message to Bob is: ", decoded_message)
Alice's message to Bob is:  Dear Bob, check this out. https://www.surveillancewatch.io/ See ya, Alice.
```

⇓

hence Alice's message decoded 🥂🎊🎉

The process would have failed if the integers involved were very much larger.

⇒ Because factoring n into its prime components p and q would be very much harder hence finding $\lambda(n)$ would be a difficulty eventually resulting into a computational complexity to solve the modular exponentiation for d that gives Bob's secret key.

Why message encoding Alice used would be insecure even if Bob's key involved larger integers?

⇒ Alice encoded her message without a padding scheme thus her message remains vulnerable to an attacker who can determine patterns in the ciphertext (same word blocks result into same ciphertext numbers) and exploit them to decode the message.